Disciplina: Projeto e Otimização de Algoritmos

Professor: Rafael Scopel

Trabalho 02

Arthur Both, Felipe Freitas e Gabriel Ferreira

Sumário

Problema 1 – Programação Dinâmica – “Crazy One Week Sprints”

1. O problema

Para uma determinada equipe de desenvolvimento, existem tarefas de alta e baixa dificuldade para serem distribuídas em sprints de 1 semana, cada uma com um valor agregado incluído. Cada semana, é necessário optar por fazer ou uma tarefa de baixa dificuldade, ou esperar uma semana para se preparar para fazer uma tarefa de alta dificuldade. Nosso objetivo é desenvolver um algoritmo que determine qual a melhor distribuição de tarefas a serem executadas para cada sprint para gerar o maior valor possível.

2. Resposta do Item 1

```
For iterations i = 1 to n
  If h_{i+1} > ℓ_i + ℓ_{i+1} then
    Output "Choose no job in week i"
    Output "Choose a high-stress job in week i+1"
    Continue with iteration i+2
  Else
    Output "Choose a low-stress job in week i"
    Continue with iteration i+1
  Endif
End
```

O algoritmo demonstrado acima não é capaz de resolver o problema completamente pois desconsidera a possibilidade de iniciar o desenvolvimento com uma tarefa de alta dificuldade, o que é uma possibilidade. Para a instância de 4 semanas, com os valores de baixa dificuldade sendo l = [10, 5, 20, 10] e os valores de alta dificuldade sendo h = [50, 5, 10, 20], o resultado esperado seria:

H(50) + L(5) + L(20) + L(10) = 85

Porém, o programa irá chegar a:

L(10) + L(5) + L(20) + L(10) = 45

Além desta, outra possível falha do algoritmo seria não considerar 2 semanas a frente, ou seja, por sempre considerar esta semana e a seguinte, pode acontecer de escolher não realizar uma tarefa e então realizar uma tarefa difícil, porém na semana seguinte a esta seria preferível não ter realizado nada na semana anterior e realizar a tarefa difícil vigente. O problema pode ser visto na instância com l = [10, 1, 10, 10], h = [5, 50, 500, 1], com resultado esperado de:

L(10) + H(0) + H(500) + L(10) = 520

O algoritmo em questão concluiria:

H(0) + H(50) + L(10) + L(10) = 70

3. Resposta do Item 2

a. O Algoritmo;

O algoritmo pensado para solucionar o problema corretamente é similar ao apresentado, porém considera iniciar com um trabalho de alta dificuldade. Iniciamos por este, consideramos as semanas seguintes e verificamos se é preferível iniciar algum trabalho ou esperar uma semana para realizar uma tarefa mais difícil na próxima semana. Finalizadas as exceções da primeira semana, consideramos agora todas as outras semanas até a penúltima, utilizando o algoritmo apresentado com uma pequena modificação; comparando sempre duas semanas com baixa dificuldade, ou uma pausa e alta dificuldade, ou ainda se é preferível uma baixa dificuldade, uma pausa e só então um trabalho de alta dificuldade, isso é, consideramos 2 semanas a frente para não acontecer de miopemente selecionar uma tarefa de alta dificuldade para a próxima semana e acabar perdendo de selecionar uma tarefa muito melhor na semana seguinte. Por fim, as duas últimas semanas são consideradas separadamente por não compararem duas semanas a frente, apenas entre si.

b. Análise do Algoritmo;

Este algoritmo tem uma complexidade de tempo de O(n), onde n é o número de semanas. Ele itera em cada semana, analisando as opções disponíveis para maximizar a receita. A análise das decisões se baseia nas receitas potenciais das semanas subsequentes, considerando as restrições impostas.

c. Implementação e Tempo de Execução;

O código implementa o algoritmo em Java. Ele realiza iterações sobre as semanas, toma decisões baseadas na maior receita possível, considerando a escolha entre tarefas de baixa e alta dificuldade e imprimirá as escolhas feitas para cada semana, junto com a receita total no final da execução. A complexidade de execução depende do número de semanas e dos cálculos necessários para determinar a melhor escolha a cada iteração. Abaixo, segue uma captura de tela da saída esperada no terminal para a instância de exemplo do professor:

	Week 1	Week 2	Week 3	Week 4
ℓ	10	1	10	10
h	5	50	5	1

```
Choose no job for week 1
Choose a high demand job for week 2
Choose a low demand job for week 3
Choose a low demand job for week 4
Total value: 70
```

Problema 2 – Branch and Bound – Knapsack

1. O problema

O problema da mochila, nesse contexto, envolve escolher a combinação mais valiosa de blocos para colocar em uma mochila com capacidade limitada de 11 kg. Cada bloco tem um valor específico e um peso associado, e o objetivo é maximizar o valor total dos blocos selecionados, garantindo que o peso total não exceda a capacidade da mochila.

2. O Algoritmo

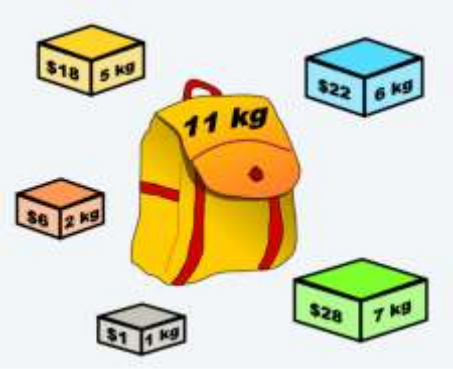
O algoritmo pensado para solucionar o problema corretamente é baseado nos princípios ensinados em aula, ele consiste em organizar os itens e pesos de acordo com a razão Valor/Peso, depois é formada uma matriz (que tem um efeito similar à árvore binária ensinada). A matriz então é preenchida com todos os valores possíveis referentes as combinações de número de itens e pesos máximos da mochila (ambos limitados ao número entregue). Por fim, após a matriz ser completamente preenchida, é utilizado um segundo algoritmo para determinar quais itens estão dentro da solução.

3. Análise do Algoritmo

Este algoritmo tem uma complexidade de tempo de $O(n \times W)$, onde n é o número total de itens e W é a capacidade máxima da mochila. Ele organiza os itens por ordem de $Valor/Peso$, itera a cada combinação de quantidade de itens com limite de peso, calculando qual o melhor valor em cada caso a fim de maximizá-lo e descobre que itens fazem parte da solução final. Os resultados dos cálculos são salvos em uma matriz, para reaproveitá-los. A análise de valor atual baseia-se nas análises realizadas em iterações anteriores, sempre considerando o número de itens e o peso máximo da mochila.

4. Implementação e Tempo de Execução

O código implementa o algoritmo em Java. Ele realiza iterações sobre as combinações possíveis de quantidades de itens (de 0 a n) com todas os limites de peso (de 0 a W) de tal modo que o valor ótimo (solução do problema) sempre estará na última posição da matriz, assim, serão impressos a nova ordem dos itens com seus valores, que itens que devem ser adicionados à mochila e o valor total desses itens na mochila. A complexidade de execução depende do número de itens, da capacidade máxima da mochila e dos cálculos necessários para determinar o valor de cada combinação de número de itens e capacidade da mochila. Abaixo, segue uma captura de tela da saída esperada no terminal para a instância de exemplo do professor:




```
Item [1]: weight = 7, value = 28, ratio = 4,00
Item [2]: weight = 6, value = 22, ratio = 3,67
Item [3]: weight = 5, value = 18, ratio = 3,60
Item [4]: weight = 2, value = 6, ratio = 3,00
Item [5]: weight = 1, value = 1, ratio = 1,00
Item 1 is not in the knapsack
Item 2 is in the knapsack
Item 3 is in the knapsack
Item 4 is not in the knapsack
Item 5 is not in the knapsack
Total value: 40
```